Forecasting Time Series Financial Data with a Recurrent Neural Network

Rizki Duwinanto (s5764971) Thomas Rey (s3977315) Evelien van Tricht (s3414019) Pedro Lavin (s5509025)

December 2023

Abstract

In this project, we work from the M3 forecasting competition dataset to try and forecast the next values in a time series. In this case, the M3 dataset contains financial data time series. We present a project where we train a simple Recurrent Neural Network (RNN) model with the time series data to estimate the best next prediction. While simpler models can sometimes achieve better results with a smart implementation, we decided on using Recurrent Neural Networks as they are built for time series prediction tasks. We present various data cleaning techniques used in the preprocessing as well as our training methods, our methods for hyperparameter tuning, and the results and finally discuss possible future implementations and reflect on our results. The final test mean squared error that the model yields is 0.0192.

Contents

1 Introduction

Financial data are important for businessmen, economists and people to know. Much financial data is generated daily—for example, the stock market, cryptocurrency, exchange rate, gross domestic product, and macroeconomy. The data generated every day indicate not only how healthy the performance of a country's economy is but also the industry inside it. These data are also used by stock brokers to buy and sell stock on the stock exchange.

The financial data generated that are recorded not only shows the past and the present financial performance but also the future because we can forecast from the recorded data. The prediction created can give an economic outlook of a country to many people and recommend data-driven decisions to businesses or governments alike. The stock exchange prediction can also benefit buyers to predict a bear or bull market and to know which stock to buy. The forecasts of financial data will benefit many if done accurately to avoid disastrous consequences such as economic crises or hyperinflation.

In this report, we try to forecast the financial data of an M3 forecast competition dataset, using a Recurrent Neural Network. M3 is a competition dataset that consists of financial data of past and present financial data. The research should give us an outlook on how financial data forecasting is done with our chosen algorithm and how can it be beneficial to be used for many people.

First, we will discuss the dataset used for this learning task in section 2. After this, we describe all the preprocessing and learning algorithm design steps in section 3. Then, in section 4, we will document the results of our learning algorithms. Lastly, section 5 will analyze said results, reflect on them and discuss directions of potential future research.

2 Data

The dataset we are using is the classic benchmark dataset for time series forecasting, the M3 forecasting competition dataset. It has been used to for forecasting competitions since 2000. This dataset is quite varied and cut up into multiple parts, 24 to be precise. It contains either yearly, monthly, quarterly and other time series. These time series are broken down into multiple categories, notably MACRO, MICRO, INDUSTRY, FINANCE, DEMOGRAPHIC and OTHER.

In this project, we work with the monthly group using the MICRO category to try to come up with a good prediction model.

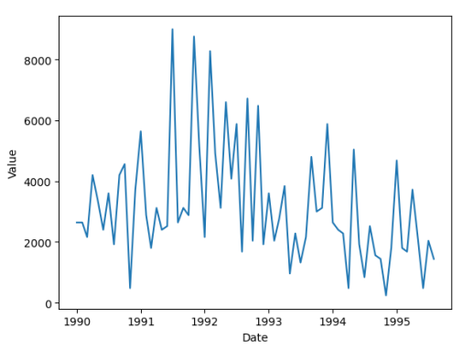

Figure 1: Example of a monthly time series.

Figure 1 shows an example of a time series. This is a random series taken from the monthly sheet starting from the first month of 1990. This data has been plotted without any cleaning, preprocessing nor detrending/deseasoning. It has been plotted only for visualization purposes.

3 Methods

3.1 Preprocessing

The M3 data is first loaded from an Excel file into Python. We want to separate our data such that we get a proper split of expected output from the input. This is done by using the last 18 data points from each time series since that is the specified resulting values as described in the dataset for the monthly time series. Then, each value in the time series is given its corresponding month and year. As the original file only indicated the initial month and year, the rest of the values were matched with the respective date.
The first part of preprocessing the dataset is to split it into training and testing. The dataset was divided between 80% for training and 20% for testing. Before this split takes place, it is necessary to prepare the data to be fed to the neural network. First, it is normalized, making every data point hold a value between -1 and 1. Normalization is done for several reasons. It makes the model handle better the existence of outliers, it makes every value more consistent, instead of letting certain points dominate over others, and normalization usually accelerates the training process.

Then, the detrending and deseasonality processes are applied. Given that the dataset spans multiple months, it exhibits implicit patterns that cyclically occur throughout the year. These fluctuations in data points do not directly correspond to shifts in financial values but rather reflect seasonal influences. Moreover, over time, there is an underlying trend causing overall value variations. To address these changes, it becomes necessary to differentiate and eliminate both the trend and seasonality components from the raw data. The detrending and deseasonalizing process is essential for isolating the actual financial trends from the extraneous temporal patterns.

This was done using the Python library STLForecast. The detrending is performed using LOESS (locally weighted regression and scatterplot smoothing) which is a method for estimating nonlinear relationships. LOESS is a technique similar to linear regression, which predicts a function for a neighbourhood of points. Each of the neighbours of a certain point x is weighted according to how far away the are, in Euclidean distance. So the linear regression equation, shown in Eq. 1

$$\beta = (XX^T)^{-1}X^TY, \tag{1}$$

where β is the vector of parameters that the linear regression model is trying to learn, X is the matrix of the independent variables of the dataset, ^{T} is the inverse of this matrix and Y is the dependent values. In LOESS, Eq. 1 is modified to Eq. 2.

$$\beta = (XWX^T)^{-1}X^TWY, \tag{2}$$

where W represents the matrix of weights. The seasonal component is then calculated by averaging out the values across different seasons.

Then, both the seasonal and trend components are subtracted from the data. The resulting values, the residual components, are left, which represent the

fluctuations without the trend and seasoning. Figure 2 shows the shape trend and seasonality elements and what the final residuals look like.

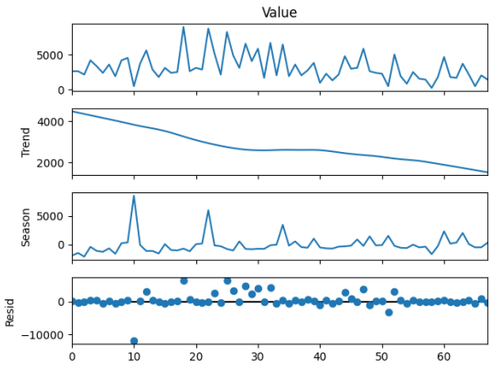

Figure 2: Example of the detrending, deseasoning and remaining residuals of one of the time series.

The next step is to prepare the input data so that it can be given to the neural network, with the appropriate input and labels. For every time series, it was divided into sets of 12 months (1 year), with the following input beginning in the immediate next month. The corresponding label was set to be the next value in the time series, as it is what the network should predict when given that input.

3.2 Learning Algorithm

3.2.1 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a type of neural network where the network has a memory form to allow outputs to be fed again into the network as inputs. This differs from an ordinary feed-forward network where data or input only flows in a single direction. Thus, data suitable to be applied to RNNs are sequential data such as time-series, sentences, and others.

Recurrent Neural Networks (RNNs) can be defined as a collection of interconnected neurons, or units, with weights defined. A unit has an activation at any given time. The model consists of K input units, each with an activation (column) vector denoted as:

$$\mathbf{u}(n) = (u_1(n), u_2(n), .., u_K(n))' \tag{3}$$

We also have N internal units, each with an activation vector denoted as:

$$\mathbf{x}(n) = (x_1(n), x_2(n), .., x_K(n))' \tag{4}$$

Finally, our model includes L output units, each with an activation vector denoted as:

$$\mathbf{y}(n) = (y_1(n), y_2(n), .., y_K(n))' \tag{5}$$

In our model, we have K equal to 1 or 1 input unit in the input layer, N equal to 3 or 3 internal units in our 1 hidden layer and L equal to 3 or 3 output units in the output layer. The unit number for the Recurrent Neural Network is defined to know the power of the network to learn from the Timeseries dataset. We also use Dropout after the input layer to avoid overfitting and we set the dropout rate to 0.2.

The input, internal and output connection weights respectively can be defined with $N \times K/N \times N/L \times (K + N)$ weight matrices with as:

$$\mathbf{W}^{in} = (w_{ij}^{in}), \mathbf{W} = (w_{ij}), \mathbf{W}^{out} = (w_{ij}^{out}) \tag{6}$$

The output units can optionally project back to internal units with links whose weights are gathered in a $N \times L$ backpropagation weight matrix.

$$\mathbf{W}^{back} = (w_{ij}^{back}) \tag{7}$$

The activation of internal units is updated according to:

$$\mathbf{x}(n+1) = f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)) \tag{8}$$

where $\mathbf{u}(n+1)$ is the input and $\mathbf{f}$ is the unit activation function. We use $\mathbf{f} = tanh$ as the activation function. The output of $\mathbf{y}(n+1)$ can be calculated according to:

$$\mathbf{y}(n+1) = f^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1))) \tag{9}$$

The activation function $tanh$ is the hyperbolic tangent unit, which can be computed as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{10}$$

Furthermore, the Recurrent Neural Network also use three sigmoid activation functions, they can be computed as:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

The loss function is introduced to measure the similarity between the input and the output of the network. Mean Squared Error is used to measure the distance between the output $\mathbf{y}(n)$ and the teacher or target vector $\hat{\mathbf{y}}(n)$. The loss function can be defined as follows:

$$\text{MSE}(y, \hat{y}) = \frac{1}{N}\sum_{u=0}^{U-1}(y_u(n) - \hat{y}_u(n))^2 \tag{12}$$

The Mean Squared Error also fits the required regression task needed to predict the output given a time series.

Our Recurrent Neural Network updates its weights with a Backpropagation Through Time (BPTT) method. To update the weight of the units in the Recurrent Neural Network, an optimization algorithm of Stochastic Gradient Descent (SGD) is also used.

The update rule for model's parameters $\mathbf{W}$ at time step t is

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \mu\nabla Q(w_t) \tag{13}$$

With ∇Q as the gradient function of the weight and μ as the learning rate. The Stochastic Gradient Descent that is used utilises momentum m_t larger than 0 and the Nesterov rule applied according to Sutskever, Martens, Dahl, and Hinton. The velocity of v_t can be described as follows:

$$v_{t+1} = m_t v_t - \mu\nabla Q(w_t + \mu v_t) \tag{14}$$

The update rule with momentum and Nesterov rule can be described as:

$$\mathbf{W}_{t+1} = \mathbf{W}_t + v_{t+1}. \tag{15}$$

3.2.2 Hyperparameter Tuning

To find the optimal hyperparameters we are solving a task of

$$\theta_{opt} = \underset{\theta \in \Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1,\ldots,N} L(\mathcal{N}_\theta(\mathbf{u}_i), \mathbf{y}_i), \tag{16}$$

where Θ is a set of all possible parameters.

There are many hyperparameters that need to be fine-tuned in order to achieve the largest accuracy, such as the number of neurons in each layer, the number of layers in the network or the learning rate. One of the most efficient ways to find these optimal hyperparameters is using cross-validation. Cross-validation is a process where the training set is divided into different subsets. The model is then trained with a subset of the training data and validated with the remaining data (the validation set). This process is then repeated multiple times varying the subsets that are used for training and validation. This makes it possible to try different values for the hyperparameters and check which one yields the best accuracy. This is of course not the definitive optimal value, as the performance depends a lot on how many subsets have been used and how different they are from the final testing data.

The cross-validation used in the time series data differs from ordinary cross-validation because we cannot shuffle the dataset as the last data is always the future data according to the time. In time series cross-validation, the training data, denoted as $X(t, t+1, .., t+n)$, with t representing time and n the length of the training dataset, is always positioned prior to the validation data. The validation data is represented as $X(t+n+1, .., X(t+n+1+m))$, where m denotes the length of the validation dataset. As we move onto the subsequent fold, the training data encompasses the validation data from the previous fold, and the test data is advanced in time by the length of the validation set plus one. This ensures that the model does not gain access to future data during the training phase, which is critical in time series forecasting. The validation split can be illustrated in Figure 3.

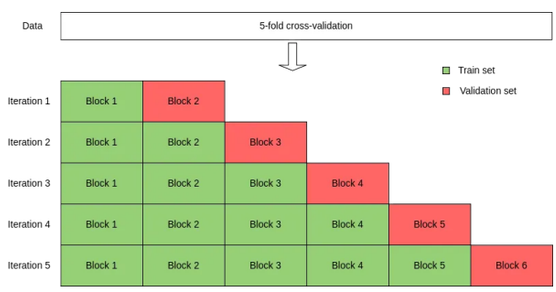

Figure 3: Illustration of 5-fold Timeseries Cross Validation Split

In our model, we made the final choice to only search for the optimal learning rate λ, leaving the rest of the hyperparameters fixed from the initial moment. The learning rate of the network, which controls how much the weights change, is a hyperparameter that could take a large range of values. A high learning rate can lead to faster convergence but might cause the model to overshoot the minimum, potentially leading to oscillations or divergence. On the other hand, a low learning rate may result in slow convergence and might get stuck in local minima. A 5-fold cross-validation was performed, setting the value of the learning rate to one of the following (0.01, 0.02, 0.05, 0.1, 0.2) and dividing the training data into 5 subsets.

After training, we see that the model with learning rate λ of 0.02 has the lowest Mean Squared Error. To get the model with the optimal parameters we trained the network with these hyperparameters for 100 epochs as that's the range where the network seems to converge.

4 Results

The network with two layers of 3 neurons each was trained on the M3 dataset with the Micro category, with the learning rate set to $\lambda = 0.02$. The model was ran for 10 epochs. The performance metric chosen for the network was MSE, as shown in Eq. 12.

Once the training was finished, the validation MSE achieved by the model before fine-tuning was 0.0179, and after fine-tuning, 0.00020, and the test MSE was 0.0192. In order to plot the results of the predictions, it was necessary to convert back the residuals to the appropriate format. The inverse transform was applied for the data points to be in the correct range, as the original time series values. Additionally, the trend and seasonality elements that were subtracted from the dataset at the preprocessing stage were added back to the prediction values. Figures 4, 5, 6 and 7 show the prediction of one of the time series of the datasets, the 473rd, in all 4 different stages of the preprocessing: only residuals, with seasonality, with trend and the final result with the complete time series, respectively.

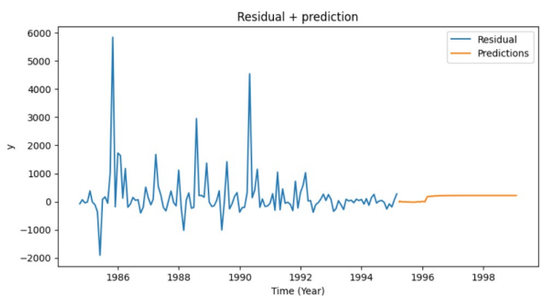

Figure 4: Figure showing the residuals of the time series and the prediction yielded by the model

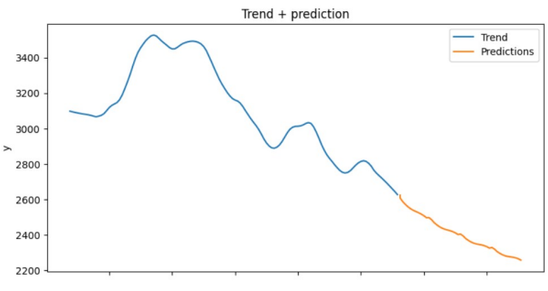

Figure 5: Graph of the trend element of one time series and its respective prediction

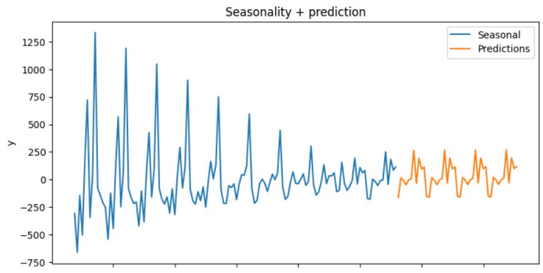

Figure 6: Graph of the seasonality element of the time series and its respective prediction

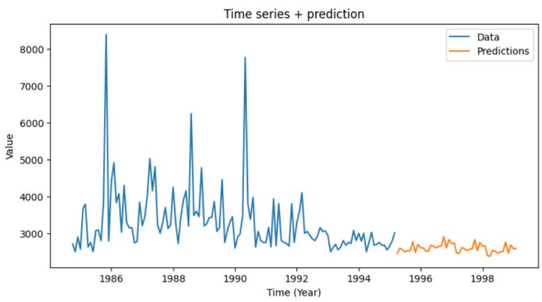

Figure 7: Graph of the complete time series and its respective prediction

5 Discussion

In this section, we will analyse our results, and the development process and talk about what worked best and what improvements could be added to the model.

5.1 Analysis

This project aimed to train a Recurrent Neural Network to learn to predict the next values in a time series as well as feed it back into the network for future predictions, this is the purpose of a Recurrent Neural Network. The main objective was not to build a model that yielded the best predictions, but rather to create a solid pipeline which preprocesses and refines the original data, fine-tunes the hyperparameters and trains a simple Recurrent Neural Network network. Even with a relatively small number of layers and units in the network (1 hidden layer with 3 neurons), the Mean Squared Error obtained is fairly low (as low as 0.0192) and the predictions are in line with what would be reasonable compared with another temporal prediction algorithm. Even though financial time series prediction is a greatly challenging task, due to the number of external factors that influence the changes and fluctuations, this report shows that with a simple network structure and consistent data preprocessing, the main patterns and overall development can be captured.

5.2 Reflection

The main challenge of designing the pipeline to forecast financial data was how to generalize a list of time series data of the same category. We have to preprocess the data to fit into the Recurrent Neural Network and it proves to be a challenging work. We also stumbled on the tools to de-season and detrend the data and also to predict the seasonality and trend after prediction as many tools do not have documentation. However, we were able to find the correct library after trial and error. The cross-validation of the Recurrent Neural Network was also broken at first because the tools or wrappers from the internet were faulty and created an infinity loop. We decided to implement from scratch ourselves late in the project.

5.3 Improvements and Future Directions

Several ways would improve the performance of the RNN to predict time series. Doing cross-validation with other hyperparameters, specifically the number of layers and neurons per layer of the network, would contribute to finding an optimal value that yields better results. Larger networks do not necessarily yield better results (Akbar, 2018), but finding a better network structure through cross-validation might. Additionally, different optimizers could be tried, such as Stochastic Gradient Descent, AdaGrad or AdaDelta. The choice of optimizer is very dependent on the type of task and dataset provided, which leaves room for finding a better one.

For machine learning tasks involving long-range dependencies and sequential data, capturing context over extended periods is crucial. GRU (Gated Recurrent Units) (Chung, Gulcehre, Cho, & Bengio, 2014) networks have been researched to improve the current Recurrent Neural Network. Also researched are LSTMs which are usually more complex than GRUs but less efficient. However, one would choose GRU for faster training time. Both networks are improved methods that could be applied to our specific task.

References

Akbar, S. (2018). Forecasting economic and financial time series: Arima vs. lstm. *arXiv*.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical evaluation of gated recurrent neural networks on sequence modeling.*

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, 17–19 Jun). On the importance of initialization and momentum in deep learning. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (Vol. 28, pp. 1139–1147). Atlanta, Georgia, USA: PMLR. Retrieved from `https://proceedings.mlr.press/v28/sutskever13.html`